

继承（Inheritance）

1.0 翻译: Hawstein (https://github.com/Hawstein) 校对: menlongsheng (https://github.com/menlongsheng)

2.0, 2.1 翻译+校对: shanks (http://codebuild.me)

2.2 校对: SketchK (https://github.com/SketchK) 2016-05-13

本页包含内容:

- 定义一个基类（Defining a Base Class）
- 子类生成（Subclassing）
- 重写（Overriding）
- 防止重写（Preventing Overrides）

一个类可以继承（*inherit*）另一个类的方法（*methods*），属性（*properties*）和其它特性。当一个类继承其它类时，继承类叫子类（*subclass*），被继承类叫超类（或父类，*superclass*）。在 Swift 中，继承是区分「类」与其它类型的一个基本特征。

在 Swift 中，类可以调用和访问超类的方法，属性和下标（*subscripts*），并且可以重写（*override*）这些方法，属性和下标来优化或修改它们的行为。Swift 会检查你的重写定义在超类中是否有匹配的定义，以此确保你的重写行为是正确的。

可以为类中继承来的属性添加属性观察器（*property observers*），这样一来，当属性值改变时，类就会被通知到。可以为任何属性添加属性观察器，无论它原本被定义为存储型属性（*stored property*）还是计算型属性（*computed property*）。

定义一个基类（Defining a Base Class）

不继承于其它类的类，称之为基类（*base class*）。

注意

Swift 中的类并不是从一个通用的基类继承而来。如果你不为你定义的类指定一个超类的话，这个类就自动成为基类。

下面的例子定义了一个叫 *Vehicle* 的基类。这个基类声明了一个名为 *currentSpeed*，默认值是 *0.0* 的存储属性（属性类型推断为 *Double*）。*currentSpeed* 属性的值被一个 *String* 类型的只读计算型属性 *description* 使用，用来创建车辆的描述。

Vehicle 基类也定义了一个名为 *makeNoise* 的方法。这个方法实际上不为 *Vehicle* 实例做任何事，但之后将会被 *Vehicle* 的子类定制：

```
class Vehicle {
    var currentSpeed = 0.0
    var description: String {
        return "traveling at \(currentSpeed) miles per hour"
    }
    func makeNoise() {
        // 什么也不做-因为车辆不一定会有噪音
    }
}
```

您可以用初始化语法创建一个 *Vehicle* 的新实例，即类名后面跟一个空括号：

```
let someVehicle = Vehicle()
```

现在已经创建了一个 *Vehicle* 的新实例，你可以访问它的 *description* 属性来打印车辆的当前速度：

```
print("Vehicle: \(someVehicle.description)")
// Vehicle: traveling at 0.0 miles per hour
```

Vehicle 类定义了一个通用特性的车辆类，实际上没什么用处。为了让它变得更加有用，需要完善它从而能够描述一个更加具体类型的车辆。

子类生成（Subclassing）

Tandem 从 Bicycle 继承了所有的属性与方法, 这又使它同时继承了 Vehicle 的所有属性与方法。Tandem 也增加了一个新的叫做 currentNumberOfPassengers 的存储型属性, 默认值为 0。

如果你创建了一个 Tandem 的实例, 你可以使用它所有的新属性和继承的属性, 还能查询从 Vehicle 继承来的只读属性 description :

```
let tandem = Tandem()  
tandem.hasBasket = true  
tandem.currentNumberOfPassengers = 2  
tandem.currentSpeed = 22.0  
print("Tandem: \(tandem.description)")  
// Tandem: traveling at 22.0 miles per hour
```

重写 (Overriding)

子类可以为继承来的实例方法 (instance method), 类方法 (class method), 实例属性 (instance property), 或下标 (subscript) 提供自己定制的实现 (implementation)。我们把这种行为叫重写 (overriding)。

如果要重写某个特性, 你需要在重写定义的前面加上 override 关键字。这么做, 你就表明了你是想提供一个重写版本, 而非错误地提供了一个相同的定义。意外的重写行为可能会导致不可预知的错误, 任何缺少 override 关键字的重写都会在编译时被诊断为错误。

override 关键字会提醒 Swift 编译器去检查该类的超类 (或其中一个父类) 是否有匹配重写版本的声明。这个检查可以确保你的重写定义是正确的。

访问超类的方法, 属性及下标

当你在子类中重写超类的方法, 属性或下标时, 有时在你的重写版本中使用已经存在的超类实现会大有裨益。比如, 你可以完善已有实现的行为, 或在一个继承来的变量中存储一个修改过的值。

在合适的地方, 你可以通过使用 super 前缀来访问超类版本的方法, 属性或下标:

- 在方法 someMethod() 的重写实现中, 可以通过 super.someMethod() 来调用超类版本的 someMethod() 方法。
- 在属性 someProperty 的 getter 或 setter 的重写实现中, 可以通过 super.someProperty 来访问超类版本的 someProperty 属性。
- 在下标的重写实现中, 可以通过 super[someIndex] 来访问超类版本中的相同下标。

极客学院

jikexueyuan.com

(http://www.jikexueyuan.com)

关于 (http://wiki.jikexueyuan.com/project/swift/)

欢迎使用 Swift (http://wiki.jikexueyuan.com/project/swift/chapter1/chapter1.html)

Swift 教程 (http://wiki.jikexueyuan.com/project/swift/chapter2/chapter2.html)

基础部分 (http://wiki.jikexueyuan.com/project/swift/chapter2/01_The_Basics.html)

基本运算符 (http://wiki.jikexueyuan.com/project/swift/chapter2/02_Basic_Operators.html)

字符串和字符 (http://wiki.jikexueyuan.com/project/swift/chapter2/03_Strings_and_Characters.html)

集合类型 (http://wiki.jikexueyuan.com/project/swift/chapter2/04_Collection_Types.html)

Via 由 [极客学院 Wiki

(http://wiki.jikexueyuan.com)]

提供

Wiki > 移动开发 > iOS > The Swift Programming Language 中文版

在子类中，你可以重写继承来的实例方法或类方法，提供一个定制或替代的方法实现。

下面的例子定义了 `Vehicle` 的一个新的子类，叫 `Train`，它重写了从 `Vehicle` 类继承来的 `makeNoise()` 方法：

```
class Train: Vehicle {
    override func makeNoise() {
        print("Choo Choo")
    }
}
```

如果你创建一个 `Train` 的新实例，并调用了它的 `makeNoise()` 方法，你就会发现 `Train` 版本的方法被调用：

```
let train = Train()
train.makeNoise()
// 打印 "Choo Choo"
```

重写属性

你可以重写继承来的实例属性或类型属性，提供自己定制的 `getter` 和 `setter`，或添加属性观察器使重写的属性可以观察属性值什么时候发生改变。

重写属性的 `Getters` 和 `Setters`

你可以提供定制的 `getter`（或 `setter`）来重写任意继承来的属性，无论继承来的属性是存储型的还是计算型的属性。子类并不知道继承来的属性是存储型的还是计算型的，它只知道继承来的属性会有一个名字和类型。你在重写一个属性时，必需将它的名字和类型都写出来。这样才能使编译器去检查你重写的属性是与超类中同名同类型的属性相匹配的。

你可以将一个继承来的只读属性重写为一个读写属性，只需要在重写版本的属性里提供 `getter` 和 `setter` 即可。但是，你不可以将一个继承来的读写属性重写为一个只读属性。

注意

如果你在重写属性中提供了 `setter`，那么你也一定要提供 `getter`。如果你不想在重写版本中的 `getter` 里修改继承来的属性值，你可以直接通过 `super.someProperty` 来返回继承来的值，其中 `someProperty` 是你想要重写的属性的名字。

以下的例子定义了一个新类，叫 `Car`，它是 `Vehicle` 的子类。这个类引入了一个新的存储型属性叫做 `gear`，默认值为整数 `1`。`Car` 类重写了继承自 `Vehicle` 的 `description` 属性，提供包含当前档位的自定义描述：

```
class Car: Vehicle {
    var gear = 1
    override var description: String {
        return super.description + " in gear \(gear)"
    }
}
```

重写的 `description` 属性首先要调用 `super.description` 返回 `Vehicle` 类的 `description` 属性。之后，`Car` 类版本的 `description` 在末尾增加了一些额外的文本来提供关于当前档位的信息。

如果你创建了 `Car` 的实例并且设置了它的 `gear` 和 `currentSpeed` 属性，你可以看到它的 `description` 返回了 `Car` 中的自定义描述：

```
let car = Car()
car.currentSpeed = 25.0
car.gear = 3
print("Car: \(car.description)")
// Car: traveling at 25.0 miles per hour in gear 3
```

重写属性观察器（Property Observer）

你可以通过重写属性为一个继承来的属性添加属性观察器。这样一来，当继承来的属性值发生改变时，你就会被通知到，无论那个属性原本是如何实现的。关于属性观察器的更多内容，请看属性观察器 ([../chapter2/10_Properties.html#property_observers](#))。

注意

你不可以为继承来的常量存储型属性或继承来的只读计算型属性添加属性观察器。这些属性的值是不可以被设置的，所以，为它们提供 `willSet` 或 `didSet` 实现是不恰当。

此外还要注意，你不可以同时提供重写的 `setter` 和重写的属性观察器。如果你想观察属性值的变化，并且你已经为那个属性提供了定制的 `setter`，那么你在 `setter` 中就可以观察到任何值变化了。

http://wiki.jikexueyuan.com/project/swift/chapter2/13_Inheritance.html

3/6

```
class AutomaticCar: Car {
    override var currentSpeed: Double {
        didSet {
            gear = Int(currentSpeed / 10.0) + 1
        }
    }
}
```

当你设置 AutomaticCar 的 currentSpeed 属性，属性的 didSet 观察器就会自动地设置 gear 属性，为新的速度选择一个合适的挡位。具体来说就是，属性观察器将新的速度值除以 10，然后向下取得最接近的整数值，最后加 1 来得到挡位 gear 的值。例如，速度为 35.0 时，挡位为 4：

```
let automatic = AutomaticCar()
automatic.currentSpeed = 35.0
print("AutomaticCar: \(automatic.description)")
// AutomaticCar: traveling at 35.0 miles per hour in gear 4
```

防止重写


你可以通过把方法，属性或下标标记为 final 来防止它们被重写，只需要在声明关键字前加上 final 修饰符即可（例如：final var，final func，final class func，以及 final subscript）。

如果你重写了带有 final 标记的方法，属性或下标，在编译时会报错。在类扩展中的方法，属性或下标也可以在扩展的定义里标记为 final 的。

你可以通过在关键字 class 前添加 final 修饰符（final class）来将整个类标记为 final 的。这样的类是不可被继承的，试图继承这样的类会导致编译报错。

上一篇: 下标 (/project/swift/chapter2/12_Subscripts.html)
下一篇: 构造过程 (/project/swift/chapter2/14_Initialization.html)

被顶起来的评论



浪子潇洞 (http://hujiaweibujidao.github.io/)


类型方法可以用static或者class来修饰，如果父类的类型方法使用static修饰的话，子类不能重写；如果父类的类型方法使用class修饰的话，子类能够重写。但是如果父类的类型方法是同时使用final class修饰的话，子类就不能重写。也就是说，static自带final性质，希望子类能够重写使用class修饰，不希望子类重写使用static修饰，final class修饰看起来别扭，因为一个允许重写（class），另一个不允许重写（final）。

代码亲测，希望没写错。😊

5月8日 回复 顶(1) 转发

24 条评论

最新 最早 最热



浪子潇洞 (http://hujiaweibujidao.github.io/)

类型方法可以用static或者class来修饰，如果父类的类型方法使用static修饰的话，子类不能重写；如果父类的类型方法使用class修饰的话，子类能够重写。但是如果父类的类型方法是同时使用final class修饰的话，子类就不能重写。也就是说，static自带final性质，希望子类能够重写使用class修饰，不希望子类重写使用static修饰，final class修饰看起来别扭，因为一个允许重写（class），另一个不允许重写（final）。

代码亲测，希望没写错。😊

5月8日 回复 顶(1) 转发



初学者

回复 Orangeince: 调用顺序应该是这样：
子类的willSet -> 父类的willSet -> 父类的didSet -> 子类的didSet

5月6日 回复 顶 转发



18610731085

重写属性观察器不好用？


4月26日 回复 顶 转发



从今以后

回复 悉毅: 可以使用协议扩展来实现类似多继承的功能

2月15日 回复 顶 转发



悉毅 (http://weibo.com/1403384842)

(http://weibo.com/1403384842)

关于 (http://wiki.jikexueyuan.com/project/swift/)

Realank刘 (http://weibo.com/realank)

回复 pml: 没错吧

(http://weibo.com/realank)1月18日 回复 顶 转发

(http://

欢迎使用 Swift (http://wiki.jikexueyuan.com/project/swift/chapter1/chapter1.html)

Orangeince

回复 xiepanqi: 调用顺序是先父类后子类吧

1月18日 回复 顶 转发

Swift 教程 (http://wiki.jikexueyuan.com/project/swift/chapter2/chapter2.html)

pml

回复 pml: 子类生成一节中：“默认情况下，你创建任何新的Bicycle实例将不会有一个篮子（即hasBasket属性默认为false），创建该实例之后，你可以为特定的Bicycle实例设置hasBasket属性为ture”-----这里最后的true写错了

1月10日 回复 顶 转发

基础部分 (http://wiki.jikexueyuan.com/project/swift/chapter2/01_The_Basics.html)

pml

定义一个基类下面的“不继承于其它类的类，称之为基类（base calss）。”-----class写错了

1月10日 回复 顶 转发

基本运算符 (http://wiki.jikexueyuan.com/project/swift/chapter2/02_Basic_Operators.html)

王学森 (http://weibo.com/imwangxuesen)

回复 从今以后: 谢谢

(http://weibo.com/imwangxuesen)2015年11月17日 回复 顶 转发

字符串和字符 (http://wiki.jikexueyuan.com/project/swift/chapter2/03_Strings_and_Characters.html)

从今以后 (http://t.qq.com/congjinyih6382)

回复 xiepanqi: 这应该是因为子类的 setter 会去进一步调用父类的 setter 吧

(http://t.qq.com/congjinyih6382)2015年11月16日 回复 顶 转发

集合类型 (http://wiki.jikexueyuan.com/project/swift/chapter2/04_Collection_Tvnes.html)

从今以后 (http://t.qq.com/congjinyih6382)

回复 王学森: 我在 playground 中试了下，train.makeNoise() 这行右侧预览窗格对应的是类名，打印语句显示在了 print("Choo Choo") 那行，底下的控制台输出是没问题的~

(http://t.qq.com/congjinyih6382)2015年11月16日 回复 顶 转发

Via 由 [极客学院 Wiki

从今以后 (http://t.qq.com/congjinyih6382)

回复 proud2008: 已经改过来了~

(http://t.qq.com/congjinyih6382)2015年11月16日 回复 顶 转发

(http://wiki.jikexueyuan.com)]

从今以后 (http://t.qq.com/congjinyih6382)

回复 马资懿: 用 final 修饰符标记的类不可被继承，单独用 final 标记的方法、属性、下标脚本不可被重写。对于类来说，static 修饰符相当于“final class”两个修饰符组合的效果，试图重写这种类方法或类型属性，会导致错误 “error: class method overrides a 'final' class method” 或 “error: class var overrides a 'final' class var”。

2015年11月16日 回复 顶 转发

提供

xiepanqi (http://www.baidu.com/p/xiepanqi)

重写属性，get和set重写后父类的get和set不会再被调用

(http://www.baidu.com/p/xiepanqi)willSet和didSet重写后父类的willSet和didSet还会继续被调用，而调用属性是先子类后父类

2015年11月16日 回复 顶 转发

王学森 (http://weibo.com/imwangxuesen)

class Train: Vehicle {

override func makeNoise() {

print("Choo Choo")

}

}

创建对象后调用 makeNoise（）不会打印choochoo 而是打印出类名字、

2015年11月13日 回复 顶 转发

马资懿 (http://weibo.com/emosad)

回复 magi: 类方法不是static 关键字修饰的么

(http://weibo.com/emosad)2015年11月4日 回复 顶 转发

markxia

回复 马资懿: 防止被继承，final 就是说这个类最终的类，不能被其它的类型继承

2015年11月3日 回复 顶 转发

magi

回复 马资懿: 不可被重写的类方法

2015年11月2日 回复 顶 转发

关于 (http://wiki.jikexueyuan.com/project/swift/)

欢迎使用 Swift (http://wiki.jikexueyuan.com/project/swift/chapter1/chapter1.html)

Swift 教程 (http://wiki.jikexueyuan.com/project/swift/chapter2/chapter2.html)

基础部分 (http://wiki.jikexueyuan.com/project/swift/chapter2/01_The_Basics.html)

基本运算符 (http://wiki.jikexueyuan.com/project/swift/chapter2/02_Basic_Operators.html)

字符串和字符 (http://wiki.jikexueyuan.com/project/swift/chapter2/03_Strings_and_Characters.html)

集合类型 (http://wiki.jikexueyuan.com/project/swift/chapter2/04_Collection_Types.html)

Via 由 [极客学院 Wiki

(http://wiki.jikexueyuan.com)]

提供

继承 (Inheritance) - The Swift Programming Language 中文版 - 极客学院Wiki



馬资懿 (http://weibo.com/emosad)

想知道endless framework 是否用于啥的?

Wiki > 移动开发 > iOS > The Swift Programming Language 中文版
(http://weibo.com/emosad)2015年10月22日 回复 顶 转发

1 2

社交帐号登录: 微信 微博 QQ 人人 更多»



说点什么吧...

发布

[极客学院 Wiki - wiki.jikexueyuan.com] 正在使用多说 (http://duoshuo.com)